2017 IEEE International Conference on Robotics and Automation (ICRA)
Singapore, May 29 - June 3, 2017

Information Theoretic MPC for Model-Based Reinforcement Learning

Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews,
James M. Rehg, Byron Boots, and Evangelos A. Theodorou

Abstract— **We introduce an information theoretic model predictive control (MPC) algorithm capable of handling complex cost criteria and general nonlinear dynamics. The generality of the approach makes it possible to use multi-layer neural networks as dynamics models, which we incorporate into our MPC algorithm in order to solve model-based reinforcement learning tasks. We test the algorithm in simulation on a cart-pole swing up and quadrotor navigation task, as well as on actual hardware in an aggressive driving task. Empirical results demonstrate that the algorithm is capable of achieving a high level of performance and does so only utilizing data collected from the system.**

I. INTRODUCTION

Many robotic tasks can be framed as reinforcement learning (RL) problems, where a robot seeks to optimize a cost function encoding a task by utilizing data collected by the system. The types of reinforcement learning problems encountered in robotic tasks are frequently in the continuous state-action space and high dimensional [1]. The methods for solving these problems are often categorized into model-free and model-based approaches.

Model-free approaches to RL, such as policy gradient methods, have been successfully applied to many challenging tasks [2]–[4] These approaches typically require an expert demonstration to initialize the learning process, followed by many interactions with the actual robotic system. Unfortunately, model-free approaches often require a large amount of data from these interactions, which limits their applicability. Additionally, while optimization of the initial demonstrated policy leads to improved task performance, in the most popular gradient-based appraches the resulting solution remains within the envelope of the initially demonstrated policy. This limits the method's ability to discover novel optimal control behaviors.

In the second paradigm, model-based RL approaches first learn a model of the system and then train a feedback control policy using the learned model [5]–[7]. Other techniques for model-based reinforcement learning incorporate trajectory optimization with model learning [8] or disturbance learning [9]. This means that interactions with the robotic system must be performed at every iteration of the trajectory optimization algorithm.

Despite all of the progress on both model-based and model-free RL methods, generalization remains a primary challenge. Robotic systems that operate in changing and stochastic environments must adapt to new situations and be equipped with fast decision making processes. Model predictive control (MPC) or receding horizon control tackles this problem by relying on online optimization of the cost function and is one of the most effective ways to achieve generalization for RL tasks [10]. However, most variations of MPC rely on tools from constrained optimization, which means that convexification (such as with quadratic approximation) of the cost function and first or second order approximations of the dynamics are required.

Fig. 1. Aggressive driving with model predictive path integral control and neural network dynamics.

A more flexible MPC method is model predictive path integral (MPPI) control, a sampling-based algorithm which can optimize for general cost criteria, convex or not [11]–[13]. However, in prior work, MPPI could only be applied to systems with control affine dynamics. In this paper, we extend the method so that it is applicable to a larger class of stochastic systems and representations. In particular, we demonstrate how the update law used in MPPI can be derived through an information theoretic framework, *without* making the control affine assumption. This is a significant step forward because it enables a purely data-driven approach to model learning within the MPPI framework. We use multi-layer neural networks to approximate the system dynamics, and demonstrate the ability of MPPI to perform difficult real time control tasks using the approximate system model. We test the MPPI controller in simulation, using purely learned neural network dynamics, on a simulated cart-pole swing-up task, and a quadrotor obstacle navigation task. Simulation results demonstrate that the controller performs comparably to an "ideal" MPPI controller, which we define as the MPPI controller which has access to the actual simulation dynamics. To further demonstrate the practicality and effectiveness of the approach, we test it on real hardware in an aggressive driving task with the Georgia Tech AutoRally platform and obtain comparable results to MPPI with a hand-designed physics-based vehicle model used in our prior work [11].

The authors are with the Institute for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA, USA. Email: gradyrw@gatech.edu

978-1-5090-4633-1/17/$31.00 ©2017 IEEE

II. Model Predictive Control

The theory of model predictive control for linear systems is well understood and has many successful applications in the process industry [14]. For nonlinear systems, MPC is an increasingly active area of research in control theory [15]. Despite the progress in terms of theory and successful applications, most prior work on MPC focuses on stabilization or trajectory tracking tasks. The key difference between classical MPC and MPC for reinforcement learning is that RL tasks have complicated objectives beyond stabilization or tracking. The complexity of the objectives in RL tasks increases the computational cost of the optimization, a major problem since optimization must occur in real time. The most tractable approach to date is receding-horizon differential dynamic programming [16], which is capable of controlling complex animated characters in realistic physics simulators, albeit using a ground truth model. The fusion of learned system models with the type of generalized MPC necessary for solving RL problems is an emerging area of research.

The information theoretic MPC algorithm that we develop is originally based on path integral control theory. In its traditional form, path integral control involves taking an exponential transform of the value function of an optimal control problem and then applying the Feynman-Kac formula to express the solution to the Hamilton-Jacobi-Bellman partial differential equation in terms of an expectation over all possible system paths. To make this transformation, the dynamics must be affine in controls and satisfy a special relationship between noise and controls. In [17], this approach was connected to the information theoretic notions of free energy and relative entropy (also known as KL divergence), which was then exploited in [11] to derive a slightly generalized path integral expression. Here, we take this one step further, and completely remove the control affine requirement. The resulting derivation and update law are closely related to the cross-entropy method for stochastic diffusion processes [18], as well as reward weighted regression [19]. However, those prior works are geared towards updating the parameters of a feedback control policy, whereas we focus on optimizing an open-loop control plan for use with MPC.

III. Information Theoretic Control

In this section we introduce the theoretical basis for our sampling based MPC algorithm. The derivation relies on two important concepts from information theory: the KL divergence and free energy. The result of the derivation is an expression for an optimal control law in terms of a weighted average over sampled trajectories. This leads to a gradient-free update law which is highly parallelizable, making it ideal for online computation.

A. Objective Function

We consider the discrete time stochastic dynamical system:

$$\mathbf{x}_{t+1} = \mathbf{F}(\mathbf{x}_t, \mathbf{v}_t) \tag{1}$$

The state vector is denoted $\mathbf{x}_t \in \mathbb{R}^n$, and $\mathbf{u}_t \in \mathbb{R}^m$ is the commanded control input to the system. We assume that if we apply an input $\mathbf{u}_t$ then the actual input will be $\mathbf{v}_t \sim \mathcal{N}(\mathbf{u}_t, \Sigma)$. This is a reasonable noise assumption for many robotic systems where the commanded input has to pass through a lower level controller. A prototypical example is the steering and throttle inputs for a car which are then used as set-point targets for low level servomotor controllers.

We define $V = (\mathbf{v}_0, \mathbf{v}_1, \dots \mathbf{v}_{T-1})$ as a sequence of inputs over some number of timesteps T. This sequence is itself a random variable defined as mapping $V : \Omega \to \Omega_V$ where Ω is the sample space and $\Omega_V = \mathbb{R}^m \times \{0, 1, \dots T-1\}$ is the image of Ω. Note that by changing the control input sequence $U = (\mathbf{u}_0, \mathbf{u}_1 \dots \mathbf{u}_{T-1})$ we can change the probability distribution for V. There are three distinct distributions which we are interested in. First, we denote $\mathbb{P}$ as the probability distribution of an input sequence in the uncontrolled system (i.e. $U \equiv 0$), next we denote $\mathbb{Q}$ as the distribution when the control input is an open-loop control sequence. Lastly, we denote $\mathbb{Q}^*$ as an abstract "optimal distribution" which we will define shortly. The probability density functions for these distributions are denoted as $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{q}^*$ respectively. Note that the density functions $\mathbf{p}$ and $\mathbf{q}$ have simple analytic forms given by:

$$\mathbf{p}(V) = \prod_{t=0}^{T-1} Z^{-1} \exp\left(-\frac{1}{2}\mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t\right) \tag{2}$$

$$\mathbf{q}(V) = \prod_{t=0}^{T-1} Z^{-1} \exp\left(-\frac{1}{2}(\mathbf{v}_t - \mathbf{u}_t)^{\mathrm{T}}\Sigma^{-1}(\mathbf{v}_t - \mathbf{u}_t)\right) \tag{3}$$

$$Z = ((2\pi)^m |\Sigma|)^{\frac{1}{2}} \tag{4}$$

Given an initial condition $\mathbf{x}_0$ and an input sequence V, we can uniquely determine the corresponding system trajectory by recursively applying $\mathbf{F}$. We thus have a mapping from inputs V to trajectories, denoted as τ. Let $\Omega_\tau \subset \mathbb{R}^n \times \{0, \dots T-1\}$ be the space of all possible trajectories and define:

$$\mathcal{G}_{\mathbf{x}_0} : \Omega_V \to \Omega_\tau \tag{5}$$

as the function which maps the input sequences to trajectories for the given initial condition $\mathbf{x}_0$. Now consider a state-dependent cost function for trajectories:

$$C(\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_T) = \phi(\mathbf{x}_T) + \sum_{t=1}^{T-1} q(\mathbf{x}_t) \tag{6}$$

where $\phi(\cdot)$ is a terminal cost and $q(\cdot)$ is an instantaneous state cost. We can use this to create a cost function over input sequences by defining $S : \Omega_V \to \mathbb{R}^+$ as the composition:

$$S = C \circ \mathcal{G}_{\mathbf{x}_0} \tag{7}$$

Now let λ be a positive scalar variable. The free-energy of the control system $(\mathbf{F}, S, \lambda)$ is defined as:

$$\mathcal{F}(V) = -\lambda \log\left(\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(V)\right)\right]\right) \tag{8}$$

Next we switch the expectation to be with respect to $\mathbb{Q}$ by adding in the likelihood ratio $\frac{\mathbf{p}(V)}{\mathbf{q}(V)}$. This yields:

$$\mathcal{F}(V) = -\lambda \log\left(\mathbb{E}_{\mathbb{Q}}\left[\frac{\mathbf{p}(V)}{\mathbf{q}(V)}\exp\left(-\frac{1}{\lambda}S(V)\right)\right]\right) \tag{9}$$

Now apply Jensen's inequality:

$$\mathcal{F}(V) \leq -\lambda \mathbb{E}_{\mathbb{Q}}\left[\log\left(\frac{\mathbf{p}(V)}{\mathbf{q}(V)}\exp\left(-\frac{1}{\lambda}S(V)\right)\right)\right] \quad (10)$$

Note that if $\frac{\mathbf{p}(V)}{\mathbf{q}(V)} \propto \frac{1}{\exp\left(-\frac{1}{\lambda}S(V)\right)}$, then the term inside the logarithm will reduce to a constant, which means that Jensen's inequality becomes an equality. This will be important shortly. Now we continue by rewriting the right-hand side of (10) as:

$$-\lambda \mathbb{E}_{\mathbb{Q}}\left[\log\left(\frac{\mathbf{p}(V)}{\mathbf{q}(V)}\right) + \log\left(\exp\left(-\frac{1}{\lambda}S(V)\right)\right)\right] \quad (11)$$

$$= \mathbb{E}_{\mathbb{Q}}\left[S(V) - \lambda \log\left(\frac{\mathbf{p}(V)}{\mathbf{q}(V)}\right)\right] \quad (12)$$

$$= \mathbb{E}_{\mathbb{Q}}\left[S(V) + \frac{\lambda}{2}\sum_{t=0}^{T-1}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t\right] \quad (13)$$

where the final step is a consequence of (2 – 3) and the fact that $\mathbf{v}_t$ has mean $\mathbf{u}_t$ under the distribution $\mathbb{Q}$. So, mirroring the approach in [17], we have the cost of an optimal control problem bounded from below by the free energy of the system. Now we want to define an "optimal distribution" for which the bound is tight. We define $\mathbb{Q}^*$ through its density function:

$$\mathbf{q}^*(V) = \frac{1}{\eta}\exp\left(-\frac{1}{\lambda}S(V)\right)\mathbf{p}(V) \quad (14)$$

where η is the normalizing constant. It is easy to see that this distribution satisfies the condition that it is proportional to $1/\exp\left(-\frac{1}{\lambda}S(V)\right)$. So Jensen's inequality reduces to an equality for $\mathbb{Q}^*$, and the bound is tight. Now that we have an optimal distribution, we can follow the approach in [11] and compute the control to push the controlled distribution as close as possible to the optimal one. This corresponds to the optimization problem:

$$U^* = \underset{U}{\operatorname{argmin}}\, \mathbb{D}_{\mathrm{KL}}\left(\mathbb{Q}^* \parallel \mathbb{Q}\right) \quad (15)$$

B. KL Divergence Minimization

Our goal is to derive an expression for the optimal controls defined in (15). Using the definition of KL divergence, we have $\mathbb{D}_{\mathrm{KL}}\left(\mathbb{Q}^* \parallel \mathbb{Q}\right)$ equal to:

$$\int_{\Omega_V} \mathbf{q}^*(V)\log\left(\frac{\mathbf{q}^*(V)}{\mathbf{q}(V)}\right)\mathrm{d}V$$

$$= \int_{\Omega_V} \mathbf{q}^*(V)\log\left(\frac{\mathbf{q}^*(V)}{\mathbf{p}(V)}\frac{\mathbf{p}(V)}{\mathbf{q}(V)}\right)\mathrm{d}V$$

$$= \int_{\Omega_V} \underbrace{\mathbf{q}^*(V)\log\left(\frac{\mathbf{q}^*(V)}{\mathbf{p}(V)}\right)}_{\text{Independent of } U} - \mathbf{q}^*(V)\log\left(\frac{\mathbf{q}(V)}{\mathbf{p}(V)}\right)\mathrm{d}V$$

Neither the optimal distribution nor the distribution corresponding to the uncontrolled dynamics depends on the control input that we apply. This can be verified by examining the density functions $\mathbf{p}$ and $\mathbf{q}^*$. Therefore, the left-most term does not depend on U and can be removed:

$$U^* = \underset{U}{\operatorname{argmax}} \int_{\Omega_V} \mathbf{q}^*(V)\log\left(\frac{\mathbf{q}(V)}{\mathbf{p}(V)}\right)\mathrm{d}V \quad (16)$$

Note that we have flipped the sign and changed the minimization to a maximization. It is easy to show that:

$$\frac{\mathbf{q}(V)}{\mathbf{p}(V)} = \exp\left(\sum_{t=0}^{T-1} -\frac{1}{2}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t\right) \quad (17)$$

Inserting this into (16) yields:

$$\int \mathbf{q}^*(V)\left(\sum_{t=0}^{T-1} -\frac{1}{2}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t\right)\mathrm{d}V \quad (18)$$

After integrating out the probability in the first term, this expands out to:

$$\sum_{t=0}^{T-1}\left(-\frac{1}{2}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + \mathbf{u}_t^{\mathrm{T}}\int \mathbf{q}^*(V)\Sigma^{-1}\mathbf{v}_t\mathrm{d}V\right) \quad (19)$$

This is concave with respect to each $\mathbf{u}_t$, so we can find the maximum with respect to each $\mathbf{u}_t$ by taking the gradient and setting it to zero. Doing this yields:

$$\mathbf{u}_t^* = \int \mathbf{q}^*(V)\mathbf{v}_t\mathrm{d}V \quad (20)$$

C. Importance Sampling

If we could sample from the optimal distribution $\mathbb{Q}^*$, then we could compute $\mathbf{u}_t^*$ by drawing samples from $\mathbb{Q}^*$ and averaging them. However, we clearly cannot sample from $\mathbb{Q}^*$, so we need to be able to compute the integral by sampling from a proposal distribution. Consider that (20) can be rewritten as:

$$\int \mathbf{q}(V)\underbrace{\frac{\mathbf{q}^*(V)}{\mathbf{p}(V)}\frac{\mathbf{p}(V)}{\mathbf{q}(V)}}_{w(V)}\mathbf{v}_t\mathrm{d}V \quad (21)$$

Therefore, the optimal controls can be rewritten as an expectation with respect to $\mathbb{Q}$:

$$\mathbf{u}_t^* = \mathbb{E}_{\mathbb{Q}}[w(V)\mathbf{v}_t] \quad (22)$$

where the importance sampling weight is:

$$w(V) = \frac{\mathbf{q}^*(V)}{\mathbf{p}(V)}\exp\left(\sum_{t=0}^{T-1} -\mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + \frac{1}{2}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t\right)$$

$$= \frac{1}{\eta}\exp\left(-\frac{1}{\lambda}S(V) + \sum_{t=0}^{T-1} -\mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + \frac{1}{2}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t\right) \quad (23)$$

Lastly, we make a change of variables $\mathbf{u}_t + \epsilon_t = \mathbf{v}_t$, and denote the noise sequence as $\mathcal{E} = (\epsilon_0, \epsilon_1 \ldots \epsilon_{T-1})$. We then define $w(\mathcal{E})$ as:

$$\frac{1}{\eta}\exp\left(-\frac{1}{\lambda}\left(S(U+\mathcal{E}) + \lambda\sum_{t=0}^{T-1}\frac{1}{2}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}(\mathbf{u}_t + 2\epsilon_t)\right)\right) \quad (24)$$

Where η can be approximated using the Monte-Carlo estimate:

$$\eta = \sum_{n=1}^{N}\exp\left(-\frac{1}{\lambda}\left(S(U+\mathcal{E}_n) + \lambda\sum_{t=0}^{T-1}\frac{1}{2}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}(\mathbf{u}_t + 2\epsilon_t^n)\right)\right) \quad (25)$$

with each of the N samples drawn from the system with U as the control input. We then have the iterative update law:

$$\mathbf{u}_t^{i+1} = \mathbf{u}_t^i + \sum_{n=1}^{N} w(\mathcal{E}_n)\epsilon_t^n \tag{26}$$

Note that the iterative procedure exists simply to improve the Monte-Carlo estimate of (21) by using a more accurate importance sampler. If we could compute (21) with zero error, then we could minimize the KL-Divergence between the controlled and optimal distribution in a single step.

The result of this optimization is that the expectation of inputs sampled from the controlled distribution is the same as the expectation of the inputs sampled from the optimal distribution ($\mathbb{E}_{\mathbb{Q}}[\mathbf{v}_t] = \mathbb{E}_{\mathbb{Q}^*}[\mathbf{v}_t]$). Under the control-affine assumptions in [20], it can be shown that this is equivalent to computing the optimal control. However, it is not known if this is true in general. In particular, if the optimal distribution is highly multi-modal, than expectation matching might be insufficient, and it may be necessary to use a more complex parameterization than an open loop sequence. However, empirical results do not seem to indicate that this is a problem, even when there are apparent multi-modalities in the solution space (e.g. navigating a field of obstacles).

IV. MPC with Neural Network Dynamics

To deploy (26) in an MPC setting, we need a model to sample from. In the model-based RL setting, this means learning a model from data. In this section, we describe our learning procedure and the real-time MPC implementation of (26).

A. Learning Neural Network Models

The kinematics for our robotic systems of interest are trivial given the velocities, so we need only learn the dynamics of each system (i.e., its acceleration). Specifically, given that the state $\mathbf{x}$ is partitioned as $\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}})$, where $\mathbf{q}$ is the configuration of the system and $\dot{\mathbf{q}}$ is its time derivative, we seek a function $\mathbf{f}$ so that the full state transition is:

$$\mathbf{x}_{t+1} = \mathbf{F}(\mathbf{x}_t, \mathbf{u}_t) = \begin{bmatrix} \mathbf{q}_t + \dot{\mathbf{q}}_t \Delta t \\ \dot{\mathbf{q}}_t + \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)\Delta t \end{bmatrix}$$

where Δt is a discrete time increment. We represent $\mathbf{f}$ with a fully-connected, multi-layer, neural network and train it on a dataset of state-action-acceleration triplets $\mathcal{D} = \{(\mathbf{x}_t, \mathbf{u}_t, (\dot{\mathbf{q}}_{t+1} - \dot{\mathbf{q}}_t)/\Delta t)\}_t$ using minibatch gradient descent with the RMSProp optimizer [21].

To create a dataset for learning the model, we follow a two-phase approach. In the first phase, we collect system identification data and then train the neural network. For simulation data, the MPPI controller with the ground truth model is run, whereas a human driver is used in real-world experiments. The ability to collect a bootstrapping dataset in this manner is one of the main benefits of model-based RL approaches: they can use data collected from *any* interaction with the system since the dynamics do not usually depend on the algorithm controlling the system or the task being performed. In the second phase, we repeatedly run the MPPI algorithm with the neural network model, augment the dataset from the system interactions, and re-train the neural network using the augmented dataset. In some cases, the initial dataset is enough to adequately perform the task. This procedure is shown in Alg. 1.

Algorithm 1: MPPI with Neural Network Training

Input: Task, N: Iterations, M: Trials per iteration
$\mathcal{D} \leftarrow$ CollectBootstrapData();
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{F} \leftarrow$ Train($\mathcal{D}$);
  for $j \leftarrow 0$ **to** M **do**
    $\mathcal{D}_j \leftarrow$ MPPI(**F**, Task) ;
    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_j$

We perform this procedure using a range of neural network sizes in order to determine the effect of network configuration on control performance. Table I describes the network configurations that we use in our simulations and experiments, all of the networks that we use are fully-connected networks with two hidden layers.

TABLE I
Layer sizes and activations of models

System	Layer 1 Size	Layer 2 Size	Activation
Cart-Pole	16	16	Tanh
Cart-Pole	32	32	Tanh
Cart-Pole	64	64	Tahh
Quadrotor	16	16	Tanh
Quadrotor	32	32	Tanh
Quadrotor	64	64	Tanh
AutoRally	32	32	Tanh

B. MPC Algorithm

In model predictive control, optimization and execution take place simultaneously: a control sequence is computed, and then the first element of the sequence is executed. This process is repeated using the un-executed portion of the previous control sequence as the importance sampling trajectory for the next iteration. In order to ensure that at least one trajectory has non-zero mass (i.e., at least one trajectory has low cost), we subtract the minimum cost of all the sampled trajectories from the cost function. Note that subtracting by a constant has no effect on the location of the minimum. The key requirement for sampling-based MPC is to produce a large number of samples in real time. As in [11], we perform sampling in parallel on a graphics processing unit (GPU) using Nvidia's CUDA architecture.

The use of neural networks as models makes sampling in real time considerably more difficult because forward propagation of the network can be expensive, and this operation must be performed $T \times K$ times. For example, the dynamics model for the autorally vehicle in [11] consisted of 100 parameters and a single matrix multiply, whereas the neural network model that we learn for the AutoRally task

Algorithm 2: MPPI

Given: $\mathbf{F}$: Transition Model;
K: Number of samples;
T: Number of timesteps;
$(\mathbf{u}_0, \mathbf{u}_1, ...\mathbf{u}_{T-1})$: Initial control sequence;
Σ, ϕ, q, λ: Control hyper-parameters;
while *task not completed* **do**
  $\mathbf{x}_0 \leftarrow$ GetStateEstimate();
  for $k \leftarrow 0$ **to** $K-1$ **do**
    $\mathbf{x} \leftarrow \mathbf{x}_0$;
    Sample $\mathcal{E}^k = \{\epsilon_0^k, \epsilon_1^k, \dots \epsilon_{T-1}^k\}$;
    for $t \leftarrow 1$ **to** T **do**
      $\mathbf{x}_t \leftarrow \mathbf{F}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1} + \epsilon_{t-1}^k)$;
      $S(\mathcal{E}^k) \mathrel{+}= \mathbf{q}(\mathbf{x}_t) + \lambda \mathbf{u}_{t-1}^{\mathrm{T}} \Sigma^{-1} \epsilon_{t-1}^k$;
    $S(\mathcal{E}^k) \mathrel{+}= \phi(\mathbf{x}_T)$;
  $\beta \leftarrow \min_k[S(\mathcal{E}^k)]$;
  $\eta \leftarrow \sum_{k=0}^{K-1} \exp\left(-\frac{1}{\lambda}(S(\mathcal{E}^k) - \beta)\right)$;
  for $k \leftarrow 0$ **to** $K-1$ **do**
    $w(\mathcal{E}^k) \leftarrow \frac{1}{\eta} \exp\left(-\frac{1}{\lambda}(S(\mathcal{E}^k) - \beta)\right)$;
  for $t \leftarrow 0$ **to** $T-1$ **do**
    $\mathbf{u}_t \mathrel{+}= \sum_{k=1}^{K} w(\mathcal{E}^k) \epsilon_t^k$;
  SendToActuators($\mathbf{u}_0$);
  for $t \leftarrow 1$ **to** $T-1$ **do**
    $\mathbf{u}_{t-1} \leftarrow \mathbf{u}_t$;
  $\mathbf{u}_{T-1} \leftarrow$ Intialize($\mathbf{u}_{T-1}$);

has 1412 parameters and consists of successive large matrix multiplications and non-linearities. To make this tractable we take advantage of the parallel nature of neural networks and further parallelize the algorithm by using multiple CUDA threads per trajectory sample.

V. Simulated Results

We test our approach on a simulated cart-pole swing-up and quadrotor navigation tasks. In these simulated scenarios, a convenient benchmark is the MPPI algorithm with access to the ground-truth model used for the simulation. This provides a metric for how much performance is lost by using an approximate model to the system.

A. Cart-Pole Swing Up

In this task, the controller has to swing and hold a pendulum upright using only actuation on the attached cart, starting with the pendulum oriented downwards. The state-dependent cost function has the form:

$$10x^2 + 500(\cos(\theta) + 1)^2 + \dot{x}^2 + 15\dot{\theta}^2 \tag{27}$$

The system noise is set at 0.9 and the temperature λ is set equal to 1. The bootstrapping dataset for the cart-pole comes from 5 minutes of multiple MPPI demonstrations using known dynamics but a different cost function for the swing-up task. These system identification trajectories show the cart-pole's behavior when the pole is upright, but they don't exhaust enough of the state-action space for the MPPI controller to act correctly immediately. The cart-pole is of low enough dimensionality that no bootstrap dataset is required to perform the task, though at the cost of more training iterations. The relative trajectory costs are shown in Fig. 2, where each iteration consists of one 10 second trial.

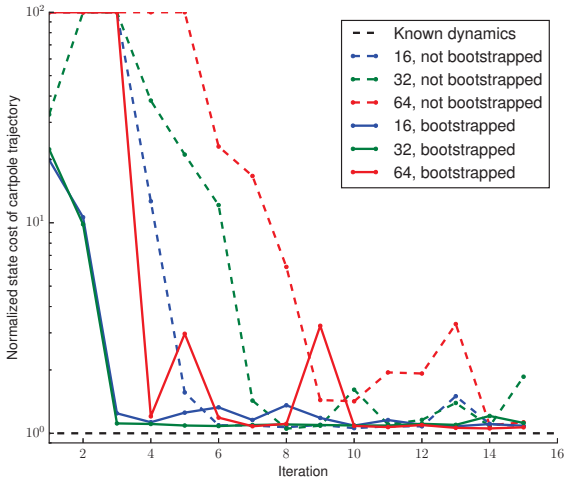

Fig. 2. Normalized state costs of executed cart-pole trajectories. The cost is normalized so that the ground-truth MPPI controller has a cost of 1. Average costs are computed from ten trials. Note the logarithmic scale and that relative costs are clamped to a maximum of 100.

B. Quadrotor Navigation

For this task, a quadrotor must fly from one corner of a field to the other while avoiding circular obstacles. We use the quadrotor model from [22], but we treat body frame angular rates and net thrust as control inputs. The state-dependent cost function has the form:

$$q(\mathbf{x}) = (\mathbf{x} - \mathbf{x}_d)^{\mathrm{T}} Q (\mathbf{x} - \mathbf{x}_d) + 100000C \tag{28}$$

$$\mathbf{x}_d = (50, 50, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0) \tag{29}$$

$$Q = \mathrm{Diag}(1, 1, 1, 25, 25, 25, 1, 1, 1, 1, 1, 1) \tag{30}$$

Here $(50, 50, 5)$ indicates the (x, y, z) position target, the variable C is an indicator variables which is turned on if the quadrotor crashes into an obstacle or the ground. The temperature was set as $\lambda = 1$ and the system noise to $(2.5, .25, .25, .25)$, where the 2.5 value corresponds to the thrust input. Since the quadrotor has twelve state and four action dimensions, bootstrapping the neural network dynamics becomes necessary. Running the algorithm without a bootstrapped neural network results in repeated failures. We bootstrap the neural network with 30 minutes of an MPPI demonstration with known dynamics and a moving target but no obstacles.

All network models yield similar results, as shown in Fig. 3. The bootstrap data is enough for the MPPI controller with the medium-sized network to navigate the field. However, the smallest and largest networks require an extra iteration to become competent at the task. After one iteration, the algorithm achieves the same level of performance regardless of which network is being used. An example trajectory successfully navigating the field is also shown in Fig. 3.

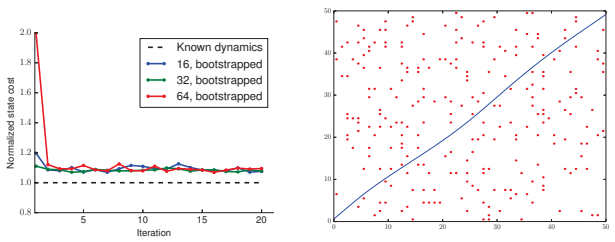

Fig. 3. Left: Normalized state costs of executed quadrotor trajectories. The costs are normalized so that the controller with the ground-truth model has a cost of 1, the cost is bounded at 2. There are five trials per iteration. Right: Example run through the virtual obstacle field, units are in meters.

C. Multi-Step Error

We train the neural network dynamics on one-step prediction error, which does not necessarily result in accurate multi-step prediction. In the worst case, compounding multi-step errors can grow exponentially [23]. The multi-step error over the prediction horizon for the cart-pole is shown in Fig. 4. For cart-pole dynamics the smaller, bootstrapped networks performed best. Note that the worst performers on multi-step error for the cart-pole directly correlate with the worst performers on the swing-up task, as one would expect.

None of the networks for the quadrotor dynamics perform significantly better or worse in multi-step error, which is reflected in the near identical performance of the MPPI controller with each of the three networks. The final positional and orientation errors after the 2.5 second prediction horizon are approximately 1.5 meters and 0.4 radians, respectively.

Mitigating the build up of model error is a primary challenge in model-based RL. MPC has two characteristics which help in this regard. The first is that it only requires a short time horizon, and the second is that it constantly recomputes the planned control sequence. The final performance margins for both the cart-pole and quadrotor are within 10% of what can be achieved with perfect model knowledge, which indicates that, in this case, our MPC algorithm is robust to these errors.

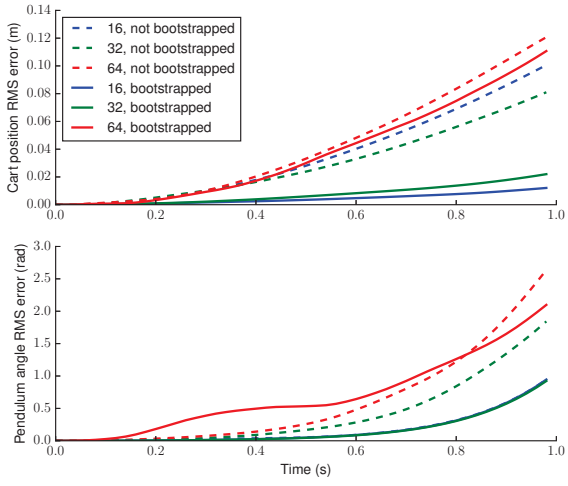

Fig. 4. Multi-step prediction error for cart position and pendulum angle.

VI. Experimental Results

We apply our approach to the task of aggressively driving around a dirt track with the Georgia Tech AutoRally vehicle from [11]. In our prior work, MPPI was successfully applied to this task using a physics-inspired model. In our current experiments, a neural network is used in place of this hand-designed model.

A. Bootstrapping Dataset

To train an initial model, we collected a system identification dataset of approximately 30 minutes of human-controlled driving at speeds varying between 4 and 10 m/s. The driving was broken into five distinct behaviors: (1) normal driving at low speeds (4–6 m/s), (2) zig-zag maneuvers performed at low speeds (4–6 m/s), (3) linear acceleration maneuvers which consist of accelerating the vehicle as much as possible in a straight line, and then braking before starting to turn, (4) sliding maneuvers, where the pilot attempts to slide the vehicle as much as possible, and (5) high speed driving where the pilot simply tries to drive the vehicle around the track as fast as possible. Each one of these maneuvers was performed for three minutes while moving around the track clockwise and for another three minutes moving counter-clockwise.

B. Experimental Setup

The experiments took place at the Georgia Tech Autonomous Racing Facility (Fig. 5). This facility consists of an elliptical dirt track approximately 3 meters wide and 30 meters across at its furthest point. The MPPI controller is provided with a global map of the track in the form of a cost-map grid. This cost-map is a smoothed occupancy grid with values of zero corresponding to the center of the track, and values of one corresponding to terrain that is not a part of the track. The cost-map grid has a 10 centimeter resolution and is stored in CUDA texture memory for efficient look-ups inside the optimization kernel. We use a neural network with 2 hidden layers of 32 neurons each and hyperbolic tangent non-linearities. The MPPI controller uses a time horizon of 2.5 seconds, a control frequency of 40 Hz, and performs 1200 samples every time-step. This corresponds to 4.8 million forward passes through the neural network every second. On-board computation is performed using an Nvidia GTX 750 Ti GPU, which has 640 CUDA cores. The cost function for the racing task had the following form:

$$q(\mathbf{x}) = 2.5(s - s_{des})^2 + 100M(x, y) + 50S_c \quad (31)$$

$$\phi(\mathbf{x}_T) = 100000C \quad (32)$$

Here s and s_{des} refer to the speed and desired speed of the vehicle, respectively. $M(x, y)$ is the cost-map value at the position (x, y), and S_c is an indicator variable which activates if the magnitude of the slip angle exceeds a certain threshold. The slip angle is defined as $-\arctan(\frac{v_y}{|v_x|})$, where v_x and v_y are the longitudinal and lateral velocities, respectively. The terminal cost is dependent on an indicator variable C which denotes whether or not the vehicle crashed

Fig. 5. Experimental setup at the Georgia Tech Autonomous Racing Facility.

at any point during the time window. During training, we set the slip angle threshold to 15.76 degrees (0.275 radians), and for the final testing runs we raised it to 21.5 degrees (0.375 radians). The sampling variance was set to 0.20 in the steering input and 0.25 in the throttle input. During training, the desired speed was set to 9 m/s (20.13 mph) and then gradually raised to 13 m/s (29.08 mph) for the final testing run. For collecting statistics, we defined a trial as 3 laps around the track starting from a full stop. Each training/test iteration consisted of three separate trial runs.

C. Results

With training settings of 9 m/s and 0.275 radians, the controller successfully maneuvered the vehicle around the track using only the initial system identification data. We performed 5 iterations, each consisting of 3 trials, for a total of 45 laps around the track. This corresponds to slightly over 8 minutes of total run-time. Adding new data into the training set and re-training the neural network model did not noticeably improve the performance of the algorithm. One explanation for this is that the initial dataset was deliberately collected for system identification, and it consists of a variety of maneuvers meant to excite various modes of the dynamics.

TABLE II

TRAINING STATISTICS

	Avg. Lap (s)	Best Lap (s)	Top Speed (m/s)	Max. Slip
Iter. 1	10.98	10.50	8.13	22.14
Iter. 2	10.79	10.32	7.84	27.4
Iter. 3	11.05	10.55	8.00	33.5
Iter. 4	10.85	10.43	7.60	25.78
Iter. 5	11.11	10.84	7.49	22.62

TABLE III

TESTING STATISTICS

	Avg. Lap (s)	Best Lap (s)	Top Speed (m/s)	Max. Slip
10 m/s	10.34	9.93	8.05	38.68
11 m/s	9.97	9.43	8.71	34.65
12 m/s	9.88	9.47	8.63	43.72
13 m/s	9.74	9.36	8.44	48.70

After the training runs, we tested the limits of the controller, using the model from the final training iteration. Specifically, we increased the threshold for penalized slip angle and the desired speed. We started with the desired speed at 10 m/s and gradually increased it to 13 m/s. At 13 m/s, the vehicle only completed two of three runs. At the lower settings, it successfully completed all trials.

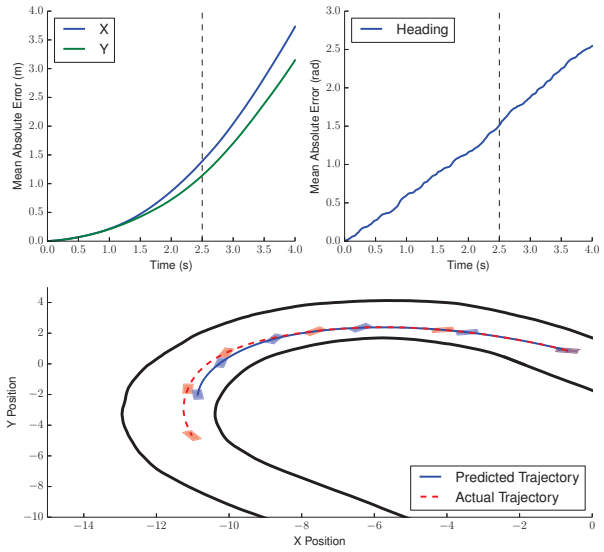

Fig. 6. Top: Multi-step prediction error on AutoRally dynamics, the vertical bar denotes the prediction horizon. Bottom: Actual trajectory vs. predicted trajectory sequence. The prediction was made off-line from an initial condition and executed control sequence that was observed while running the MPC algorithm. Orientation markers are evenly spaced in time.

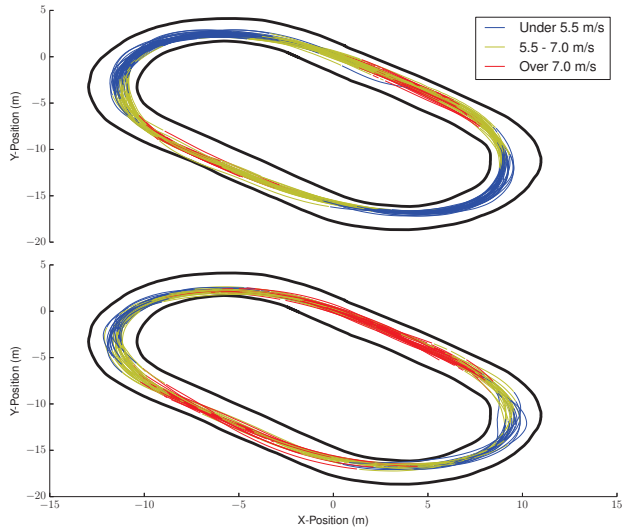

Fig. 7. Trajectory traces and speeds during training runs (top) and aggressive testing runs (bottom). Counter-clockwise travel direction.

Figure 7 shows the paths taken by the controller along with their velocity profiles. In both cases, the vehicle slows down coming into turns and accelerates out of them, eventually reaching a high speed along the middle of the straight. This is an intuitively obvious strategy, however it is worth noting that there is no portion of the cost function which explicitly enforces this behavior. Rather, this behavior emerges from the interaction between the neural network dynamics and the cost function.

Tables II and III show the statistics for the training and testing runs. The more aggressive runs complete the trials in faster times and obtain a higher top speed and maximum slip angle than the training runs. The velocity profiles of the training and test trials also differ dramatically. In the conservative training runs, the vehicle hits its top speed in the first half of the straight and immediately slows down, eventually coasting into the corner. In the aggressive setting, the vehicle maintains its speed all the way into the corner and then power slides through the turn (Fig. 1). This is demonstrated by the high slip angles in the more aggressive runs. The overall lap times achieved are slightly faster than what was achieved previously in [11]. The fastest three lap set is displayed in the accompanying video for this paper.

VII. Conclusion

We have derived an information theoretic version of model predictive path integral control which generalizes previous interpretations by allowing for non-affine dynamics. We exploited this generalization by applying the MPPI algorithm in the context of model-based reinforcement learning and used multi-layer neural networks to learn a dynamics model. In two challenging simulation tasks, the controller with the learned neural network model achieves performance within 10% of what is obtained with a perfect ground-truth model.

The scalability and practicality of the algorithm was demonstrated on real hardware in an aggressive driving scenario, where the algorithm was able to race a one-fifth scale rally car around a 30 meter long track at speeds over 8 m/s. In doing this, it performed difficult controlled power-slides around corners. This success came despite the presence of significant disturbances, such as deep grooves on portions of the track and patches of very fine loose dirt which could have easily caused the vehicle to lose traction.

This type of model-based reinforcement learning that we propose, combining generalized model predictive control with machine learning approaches for learning dynamics, is a promising new direction for solving the challenging problems that arise in robotics. The key tools in this approach are the information theoretic concepts of free energy and the KL divergence, scalable machine learning algorithms for learning dynamics, and intensive parallel computation for online optimization. The result of our approach is the emergence of complex behaviors, such as power-sliding through turns when racing, that arise purely due to the interaction between the optimization, cost function, and learned dynamics.

ACKNOWLEDGEMENTS

This work was made possible by the ARO through MURI award W911NF-11-1-0046, DURIP award W911NF-12-1-0377, NSF award NRI-1426945, and supported by the NSF Graduate Research Fellowship under Grant No. 2015207631.

References

[1] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey," in *Reinforcement Learning*. Springer, 2012, pp. 579–610.

[2] H. Benbrahim and J. A. Franklin, "Biped dynamic walking using reinforcement learning," *Robotics and Autonomous Systems*, vol. 22, no. 3, pp. 283–302, 1997.

[3] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 2219–2225.

[4] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems 21*, 2009, pp. 849–856.

[5] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[6] S. Schaal, "Learning from demonstration," in *Advances in Neural Information Processing Systems 9*, 1997, pp. 1040–1046.

[7] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, 1997, pp. 3557–3564.

[8] D. Mitrovic, S. Klanke, and S. Vijayakumar, "Adaptive optimal feedback control with learned internal dynamics models," in *From Motor Learning to Interaction Learning in Robots*. Springer, 2010, pp. 65–84.

[9] J. Morimoto and C. Atkeson, "Minimax differential dynamic programming: An application to robust biped walking," in *In Advances in Neural Information Processing Systems 15*, 2002.

[10] I. Lenz, R. Knepper, and A. Saxena, "DeepMPC: Learning deep latent features for model predictive control," in *Robotics Science and Systems*, 2015.

[11] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1433–1440.

[12] V. Gómez, S. Thijssen, A. Symington, S. Hailes, and H. J. Kappen, "Real-Time stochastic optimal control for multi-agent quadrotor systems," in *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS'16)*, 2016.

[13] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40:2, pp. 344–357, 2017.

[14] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control engineering practice*, vol. 11, no. 7, pp. 733–764, 2003.

[15] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[16] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, "An integrated system for real-time model predictive control of humanoid robots," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, 2013, pp. 292–299.

[17] E. A. Theodorou and E. Todorov, "Relative entropy and free energy dualities: Connections to path integral and KL control," in *IEEE 51st Annual Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 1466–1473.

[18] W. Zhang, H. Wang, C. Hartmann, M. Weber, and C. Schutte, "Applications of the cross-entropy method to importance sampling and optimal control of diffusions," *SIAM Journal on Scientific Computing*, vol. 36, no. 6, pp. A2654–A2672, 2014.

[19] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 745–750.

[20] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *The Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.

[21] T. Tieleman and G. Hinton, "Lecture 6.5 - RMSProp: Divide the gradient by a running average of its recent magnitude," 2012, COURSERA: Neural Networks for Machine Learning.

[22] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-uav testbed," *IEEE Robotics & Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.

[23] A. Venkatraman, M. Hebert, and J. A. Bagnell, "Improving multi-step prediction of learned time series models," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.